USING SIMULATION AS A PROXY FOR A REAL SHOP FLOOR AND DATA COLLECTION SYSTEM.

Introduction

Increasing shop floor efficiency has always been a major concern of manufacturing companies. Scheduling applications have been successfully employed to create optimized schedules that manufacturers can use to run their operations. But it is a common occurrence in today's dynamic manufacturing environment that the information upon which a schedule is based changes after the schedule has been created. Running shop floor operations based on this out-of-date information can lead to increased shop floor inefficiency. Systems need to be developed which not only provide good initial schedules, but also create new schedules in reaction to important changes in the state of operations of the shop floor.

Research is underway at the National Institute of Standards and Technology (NIST) to develop standards and procedures that facilitate the development of reactive scheduling systems [1]. A reactive scheduling system is one that not only produces initial schedules for production, but also: 1) monitors the shop floor events and evaluates compliance to the schedule and other shop floor performance related criteria; 2) collects shop floor status information to enable the creation of updated schedules based on current information; and 3) produces new schedules based on the collected information.

A key area of research in developing this approach is the use of a discrete event simulation system as a proxy for a real shop floor and a real shop floor data collection system. A prototype is being developed at NIST that uses Deneb Quest™ in these roles[*]. In this paper, the characteristics of the Shop Floor and Data Collection component of a reactive scheduling system and the modifications that were made to Quest™ to incorporate it into the NIST reactive scheduling system prototype are discussed.

Reactive Scheduling System Conceptual Overview

The goals of a reactive scheduling system (RSS) may be characterized as: to produce a feasible initial schedule that is suitable to be used in production; to monitor shop floor status as it changes during the course of production; to use current shop floor status to assess the effectiveness of the schedule; and to react to changes in shop floor environment by producing an improved schedule based on current shop floor status. Figure 1 shows the components and the inter-component data flow of such a system. A brief discussion of the functional

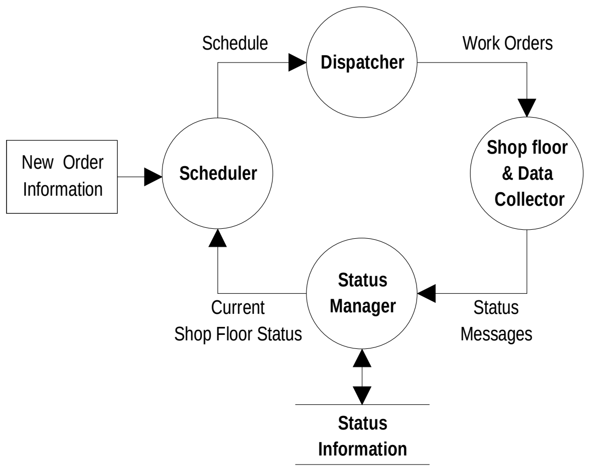

FIGURE 1: REACTIVE SCHEDULING SYSTEM COMPONENTS

[*] See end of paper for disclaimer

responsibilities of each component follows.

The Scheduler takes in order information and produces a schedule to be executed on the shop floor. The Dispatcher repackages the schedule into chronological lists of production orders for each resource and manages the delivery of those packages to the appropriate resource. The Shop floor contains machines that execute the production orders to process loads into finished products. The Shop Floor contains a Data Collector that gathers and reports shop floor status information about loads and machines in the form of status messages [2]. The Status Manager receives the status messages, stores the status information in a database, and provides an interface for accessing current status information from the database. The Status Manager may also monitor shop floor performance and request that the Scheduler perform a reschedule based on performance metrics.

A more detailed look at the Shop Floor component of the reactive scheduling system is presented in the next section.

Shop Floor Component Conceptual Overview

The shop floor is the means by which raw materials are turned into finished products. The two key entities of a shop floor are Loads and Machine Resources. A Load represents a single part or group of parts that are operated on by Machine Resources. Loads travel from one Machine Resource to another and transition from a less-finished state to a more-finished state. Information is associated with each Load that relates to its current state of processing. The status information for Loads includes an identifier to differentiate Loads from each other, the initial number of parts in a Load, the date the Load is supposed to be finished, and an identification of the product that will be produced at the end of all processing. A routing is also associated with each Load. A routing is an ordered list of the processing steps (usually called jobsteps) that the Load will go through to transition to a finished product. Each element of the list contains a jobstep name, the Machine Resource Family that is to perform the processing, and the estimated time to perform that processing.

Machine Resources are the shop floor entities that transform Loads from one state to another. As with Loads, each Machine Resource has information associated with it related to its current state of processing. The status information for Machine Resources includes an identifier to differentiate one Machine Resource from another, the identifier of the Load currently being processed by the Machine Resource (if any), and whether the Machine Resource is currently busy, idle, or broken. Also, Machine Resources are organized into Machine Resource Families. These are groups of Machine Resources with identical capabilities that operate logically as one Machine Resource. The processing capacity of a Machine Resource Family increases as the number of Machine Resources in the Machine Resource Family increases. Also associated with each Machine Resource is a dispatch list. This is an ordered-list of the work that the associated Machine Resource has been scheduled to do. Each element in this list contains an identifier of the Load to be processed (Load_id) and the name of the jobstep that is to be performed. The Dispatcher component of a RSS uses the schedule to create a dispatch list for each Machine Resource.

Events that happen on the Shop Floor as a consequence of production are collected by the Data Collector. The Data Collector uses this information to create status messages that are sent to the Status Manager component of the RSS. The Status Manager uses these messages to maintain a current view of the status of the shop floor that the scheduler can extract to produce an updated schedule.

Since Quest™ also implements something called a resource, in the rest of this paper Machine Resources will be referred to as Machines and Machine Resource Families will be referred to as Machine Families to minimize possible confusion.

In this section a conceptual overview of the Shop Floor component of a RSS was presented. In the following section the default behavior of a Quest™ simulation will be contrasted with the behavior required for a Shop Floor component of a RSS.

Quest™ simulation default behavior Vs required Shop Floor behavior

Quest™ provides a powerful system for modeling and simulating shop floor operations. With Quest™, the simulation model executes based on widget and resource definitions and processing logic encoded in the model. Some of the key features of Quest™'s default execution behavior are:

1. A source creates widgets based on the specified inter-arrival time and routes them to available workcells, buffers, or sinks that are connected to the source's outputs.
2. A buffer receives widgets from its inputs, processes them, queues them, and routes them to the workcells, buffers, or sinks that are connected to the buffer's outputs.
3. A workcell receives widgets from its inputs, process them, and routes them to the workcells, buffers, or sinks that are connected to the workcell's outputs.
4. A sink receives widgets from its inputs and destroys them.
5. Widgets get moved from the output of the current resource to the input of another resources as determined by the specified routing logic of the current resource.

In a RSS, schedule and routing information drive the execution of the shop floor. Schedule information is used to determine when loads are created and when they are to be processed by each resource. Routing information is used to determine the sequence of steps that are used to produced a finished part. Status messages related to current shop floor operations are produced by the Data Collector so that updated schedules can be produced. To have a Quest™ simulation serve as a proxy for a RSS Shop Floor, the default behavior of Quest™ must be significantly modified. The following is a brief overview of the required modifications:

1. Quest™ widgets will be used to represent Loads and Quest™ workcells will be used to represent Machines. Each widget and workcell will be augmented with user attributes to store current processing information.
2. Schedule information will determine when Loads are created and the characteristics of each load. An external Routing File will determine the sequence of processing steps for each Load at each Machine.
3. A buffer will act as an input queue for a Machine or Machine Family. A Load will remain in a buffer until a Machine requests it.
4. Each Machine will use a dispatch list to determine which Load to process next. When all Loads in a Machine's dispatch list have been processed, the Machine will become idle.
5. A Machine will search through the list of Loads contained in the buffer that is serving as its input queue, and if a Load in the buffer matches the Load specified in the dispatch list, the Machine will "request " that the Load be routed to it.

6. Machines will use routing information to determine how to process a Load and where to send the Load when processing is done.
7. A Quest™ resource will be set up as a “transfer agent” to control the movement of widgets through the system. The transfer agent will have an output that is connected to the input of each buffer that acts as an input queue for a Machine or Machine Family, and an input connected output of each Machine and source in the model.

These modifications involve defining and providing access to new kinds of data, extending the definitions of Quest™ widgets and resources, and modifying the processing and routing logic for Quest™ resources. Quest™ allows such changes to be made through its Simulation Control Language (SCL) interface. In the following sections, these SCL coding changes will be described.

Modifications to use Quest™ as a proxy

The following sections give a more detailed account of the modifications made to Quest™. Most of the changes involved a combination of SCL coding and specification of their use through the Quest™ graphical user interface (GUI). Where appropriate, issues relating to the implementation approach chosen will be discussed.

Use widgets to represent Loads

Quest™ widgets are the obvious choice to serve as Loads. Each widget was augmented with the following user attributes to facilitate its use as a Load: Load_id, Product_id, Start_amount, Current_amount, Pieces_complete, Current_jobstep, Next_jobstep, Next_resource, & Route_number. The user attribute Route_number was added to facilitate the routing of a Load to a specific Machine. Its use will be explained in a later section.

A key decision in the use of widgets as Loads is whether a different widget type should represent a Load at each step in its processing. It was determined that it is sufficient to use one widget type for all Loads. The specific kind of Load and its current state of finishing can be determined at any step in its processing by its Product_id and Current_jobstep user attributes. The downside of this approach is that the ability to gather statistics based on widget type and to assign a different shape to each widget type to indicate different levels of finishing are lost.

Use workcells to represent Machines

Just as Quest™ widgets are similar to Loads, Quest™ workcells are the obvious choice to serve as Machines. Quest workcells have several fields, such as name and status, that can be used when the workcell serves as a Machine. To facilitate its use as a Machine, each workcell was augmented with the following user attributes: Family_name & Machine_route. The Family_name user attribute identifies the Machine Family that the Machine belongs to. The Machine_route user attribute holds the output number on which the workcell is connected for the buffer that is being used as the workcell’s input queue.

Use buffers as input queues for Machines and Machine Families

Quest™ workcells have an input queue. But the facilities that are provided to control a workcell’s input queue do not provide the level of control that is necessary to have a workcell operate as a Machine in a RSS. One behavioral change that was required was to have the Data Collector send a status message when a Load arrived at a Machine. If the workcell was currently processing another widget, the recently arrived widget could not cause a message to be sent if it went directly to the workcell's input queue. By having a buffer act as a workcell's input queue, it was possible to modify the buffer's logic to cause the message to be sent at the appropriate time.

Another reason for using buffers as input queues is Machine Families. All Machines in a Machine Family use a single buffer as their input queue. Loads for all of the Machines in the Machine Family pass through this buffer. When a reschedule is done, Loads that are still unprocessed and in this "family" input queue might be reassigned to different Machines in the Machine Family than the original schedule indicated. As with the status reporting requirement mentioned earlier, a Quest™ construct external to a workcell is needed to implement this behavior. Quest™ buffers were used to provide this family input queue capability for Machine Families. Also note that all Machines belong to a Machine Family. Sometimes the number of Machines in the family is one. Therefore, it is advantageous for Quest™ buffers to be used to provide an input queue capability for individual Machines as well as Machine Families.

As with Quest™ workcells, each Quest™ buffer uses the user attribute Family_name to indicate the Machine Family to which it belongs.

Define a routing table interface

In an RSS, each Load must have a routing associated with it that indicates how the Load is to be processed by the shop floor. This implementation of the RSS uses a routing table to provide this capability. Each row indicates the product being produced, the jobstep name, the Resource Family being used for the jobstep, and processing time information for this jobstep. The jobsteps for each product are organized in the order that they must be performed. Global access procedures are provided that allow the information about each jobstep to be sequentially extracted. When a Machine begins processing a Load, it uses the access procedures to extract the routing and processing information. This information facilitates the proper processing of the Load and the proper routing of the Load to its next destination.

Define a dispatch list/arrival list interface

The dispatch list specifies the sequence of Loads to be processed by a Machine. Each Machine has its own dispatch list. The dispatch list for each Machine is stored in a file with the Machine's name. Each element of a dispatch list has an identifier for the Load to be processed and the name of the jobstep. Access procedures are provided to allow a Machine to either read or delete the top element of the list.

An arrival list is a special kind of dispatch list. Each element in this list contains information about each new Load that is to be created, including the time that the Load is to be created. The arrival list is used by a Quest™ source to create Loads. Access procedures are provided to allow a Quest™ source to either read or delete the top element in the arrival list.

Define a Data Collector interface

The Data Collector is implemented as a set of procedures that write string messages to status files. Status files are text files, where each file has a common root name suffixed with a sequence number. Each line of a status file contains one status message. As messages are generated in the simulation, the procedures write them to a temporary file. When the temporary file reaches a preset length, a status file with a sequence number higher than any existing status file is created, and the messages are copied to that status file. This approach is taken to eliminate possible file contention problems. Other RSS components that want to receive messages from the shop floor component read and delete information from the sequenced status files using normal operating system capabilities.

Use init logic to load global data and access routines

Data for which common access is required is defined in the global section of an SCL file. The variables, constants, and arrays to hold the routing data are defined in this file. Procedures for loading and accessing global data, for initializing the user attribute fields of resources, and for accessing the Data Collector functionality are also defined here. Several utility procedures and testing procedures are defined here. An initialization procedure, which calls the other procedures necessary to initialize the model, is also defined here. This initialization procedure is specified as the init procedure through the user interface.

Sources use an arrival list to determine Load creation time

In a typical Quest™ model, inter-arrival time and widget percentage information is used to determine when a source creates widgets and what kinds of widgets are created. In a RSS, the schedule determines when Loads are created and what kinds of Loads are created. To implement this behavior, the processing logic of the source must be modified to get creation information from an external arrival list. Each entry in this list contains the information necessary to create a widget and fill in the user attributes that allow a widget to be used as a Load. This information also includes the time that the Load is to be created. The elements in the arrival list are ordered chronologically according to the time specified for widget creation.

The source's processing logic starts by getting Load creation information for the next Load to be created from the arrival list. If the arrival list is empty, the source idles itself indefinitely. This can be accomplished by waiting on a change in a global variable or waiting on a signal. If it does find an arrival list entry, the source gets the Load creation time and waits until that time arrives. At that time the widget is created and updated with the Load creation information. The source then uses the product information of the Load to get the first entry in the routing table for products of this type. This routing information indicates the first Machine to process the Load. The widget is updated with this information and is put on the output of the source for routing processing. The source then deletes the entry from the dispatch list, and starts processing the next entry in the arrival list.

Machines use dispatch lists to determine the sequence of processing

In an RSS, a dispatch list determines the order in which Loads are processed by Machines. To implement this behavior in Quest™, both buffer and workcell logic must be modified.

First, each workcell must have a buffer attached to its only input, which acts as an extension of the workcell's input queue. When a widget arrives, the buffer's processing logic updates the Current_jobstep user attribute, zeroes the Route_number user attribute, and puts the widget on its output. The buffer's route logic continually checks the Route_number user attribute of all widgets on its output. Any widget with a non-zero Route_number is routed to the workcell on that output.

The workcell's processing logic starts by getting the identification of the Load and jobstep that it is to process next from the dispatch list. If the dispatch list is empty, the workcell idles itself indefinitely. This can be accomplished by waiting on a change in a global variable or waiting on a signal. If a dispatch list entry is found, the workcell searches through the list of the widgets in its attached buffer. If it does not find a widget with matching Load and jobstep information, the workcell will periodically recheck the list of widgets in the buffer for a match. When it finds a widget with Load and jobstep information that matches the information from the dispatch list entry, the workcell sets the Route_number of that widget to the number of the output on which it is connected to the buffer. This causes the route logic of the buffer to send the widget to the workcell. The workcell then deletes the entry from the dispatch list and processes the widget.

The same workcell processing logic is used when multiple workcells are connected to one buffer in a Machine Family configuration.

Machines use the routing table to determine processing characteristics

Once a widget starts processing at a workcell, the Load and jobstep information attached to the widget are used to extract routing and processing information from the routing table. The workcell uses this information to determine how long to process the widget and to modify the widget to indicate the next resource that is to process the widget and the next jobstep to be done. After completion of the processing, the widget is routed to the next resource.

Invisible transfer agent and the routing table collaborate to move loads through the system

A Quest™ resource called a transfer agent facilitates the routing of Loads through the system. This resource has an input connected to the output of each workcell and has an output connected to the input buffer of all workcells in the simulation. When a widget arrives at this resource, its processing logic takes no processing time and just passes the widget to the resource's output. The routing logic searches the list of buffers connected on its output for a buffer that has a Family_name user attribute that matches the Next_resource user attribute of the widget. The widget is then routed to the buffer with the matching information.

The transfer agent is invisible in the model because it is just a convenient mechanism for building models and routing widgets. With the transfer agent, adding a new Machine Resource only involves: 1) connecting an output of the transfer agent to the input of the buffer serving as the input queue for the Machine; and 2) connecting an input of the transfer agent to the output of the Machine's workcell. Because the routing of widgets through the system is specified in the routing table and widgets are updated with the routing information as they travel through the system, the routing agent facilitates the modeling of many different routing configurations without changing the Quest™ model or the SCL code. Only the routing table needs to be changed to implement a new routing.

A typical widget processing scenario

Following is a short description of how a widget is created and is processed in a Quest™ simulation modified to behave as a Shop Floor component of an RSS. This will show how the changes described in the previous sections implement the required behavioral changes.

1. The source reads the Load creation information from the arrival file, creates the widget according to the specified information, adds the appropriate routing information, and sends the widget to the transfer agent.

2. The transfer agent transfers the widget based on the widget's Next_resource user attribute.
3. The widget arrives at the input buffer of the indicated workcell and waits to be routed to a workcell for processing.
4. The workcell reads its dispatch list to determine which Load to process next. It looks through the list of widgets in its input buffer and modifies the routing information on the widget with information matching that extracted from the dispatch list. This causes that widget to be transferred to the workcell.
5. The buffer transfers the widget to the workcell.
6. The workcell receives the widget, gets routing and processing information from the routing table, and processes the widget. After finishing its processing, the widget is modified so that it will be routed to the next resource that is to process it, and sends the widget to the transfer agent.
7. The transfers agent transfers the widget based on the widgets Next_resource user attribute. When the widget has completed all of processing, the Next_resource field causes the widget to be sent to a sink where the widget is destroyed. Otherwise, steps 2 through 7 are repeated.

Summary

In this paper, modifications that allow a Quest™ simulation to act as a proxy for a shop floor and data collection system have been described. Through SCL coding, incorporation of routing information, and implementation of the transfer agent, Quest™ provides the ability to simulate many different shop floor configurations and to drive them with schedule information.

Work described in this paper was sponsored by the US Navy Manufacturing Technology program and the NIST Systems Integration of Manufacturing Applications program. Certain commercial software and hardware products are identified in this paper. This does not imply approval of endorsement by NIST, nor does it imply that the identified products are necessarily the best available for the purpose.

References

[1] Jones, A., Riddick, F., and Rabelo, L., Development of a Predictive/Reactive Scheduler using Genetic Algorithms and Simulation-based Scheduling Software, *Proceedings of AMPST'96*, 589-598, 1996.
[2]Riddick, F., and Loreau, A., Models For Integrating Scheduling And Shop Floor Data Collection Systems, Proceedings of IASTED MIC'97, 276-279, 1997.